回顾Hibernate：

1. hibernate开发环境搭建

----》 引入jar： hibernate.jar + required + jpa + 驱动包

---》 hibernate.cfg.xml

---》 javabean/\*.hbm.xml

---》 Application 测试

2.hibernate api

----》 Configuration

----》 SessionFactory

----》 Session

---》 Transaction

---》Query

---》Criteria

---》SQLQuery

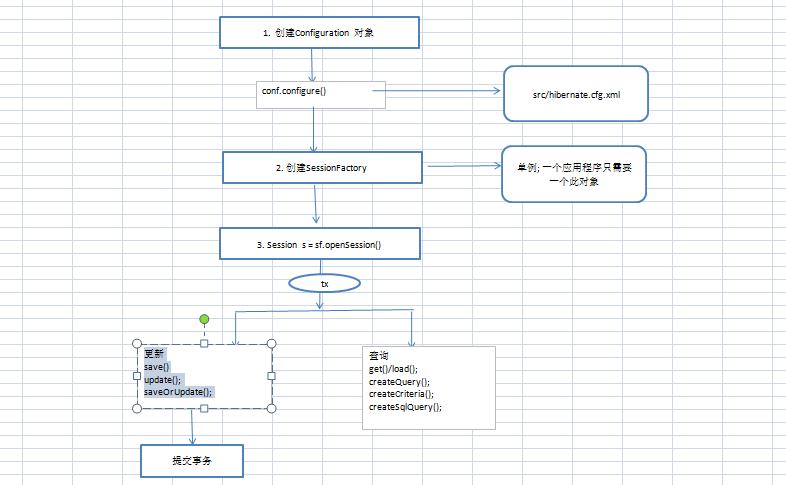
3.配置

---》 主配置

---》映射配置

\*.hbm.xml

4. hibernate 程序执行流程



目标：

关联映射(hibernate映射)

1. 集合映射

2. 一对多与多对一映射 (重点)

3. 多对多映射

4. inverse/lazy/cascade

# 1. 集合映射

开发流程：

需求分析/数据库设计、项目设计/ 编码/测试/实施部署上线/验收

需求:

用户购买， 填写地址！

数据库：

代码：

|  |
| --- |
| // javabean设计  **public** **class** User {  **private** **int** userId;  **private** String userName;  // 一个用户，对应的多个地址  **private** Set<String> address;  **private** List<String> addressList = **new** ArrayList<String>();  //private String[] addressArray; // 映射方式和list一样 <array name=""></array>  **private** Map<String,String> addressMap = **new** HashMap<String, String>();    } |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package=*"cn.itcast.a\_collection"*>    <class name=*"User"* table=*"t\_user"*>  <id name=*"userId"* column=*"id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"userName"*></property>    <!--  set集合属性的映射  name 指定要映射的set集合的属性  table 集合属性要映射到的表  key 指定集合表(t\_address)的外键字段  element 指定集合表的其他字段  type 元素类型，一定要指定  -->  <set name=*"address"* table=*"t\_address"*>  <key column=*"uid"*></key>  <element column=*"address"* type=*"string"*></element>  </set>    <!--  list集合映射  list-index 指定的是排序列的名称 (因为要保证list集合的有序)  -->  <list name=*"addressList"* table=*"t\_addressList"*>  <key column=*"uid"*></key>  <list-index column=*"idx"*></list-index>  <element column=*"address"* type=*"string"*></element>  </list>    <!--  map集合的映射  key 指定外键字段  map-key 指定map的key  element 指定map的value  -->  <map name=*"addressMap"* table=*"t\_addressMap"*>  <key column=*"uid"*></key>  <map-key column=*"shortName"* type=*"string"* ></map-key>  <element column=*"address"* type=*"string"* ></element>  </map>      </class>    </hibernate-mapping> |
| // 保存set  @Test  **public** **void** testSaveSet() **throws** Exception {  Session session = *sf*.openSession();  session.beginTransaction();    //-- 保存  Set<String> addressSet = **new** HashSet<String>();  addressSet.add("广州");  addressSet.add("深圳");  // 用户对象  User user = **new** User();  user.setUserName("Jack");  user.setAddress(addressSet);    // 保存  session.save(user);    session.getTransaction().commit();  session.close();  }    // 保存list/map  @Test  **public** **void** testSaveList() **throws** Exception {  Session session = *sf*.openSession();  session.beginTransaction();  User user = **new** User();  user.setUserName("Tom");  // // 用户对象 -- list  // user.getAddressList().add("广州");  // user.getAddressList().add("深圳");  // // 保存  // session.save(user);    // 用户对象 -- Map  user.getAddressMap().put("A0001", "广州");  user.getAddressMap().put("A0002", "深圳");    // 保存  session.save(user);    session.getTransaction().commit();  session.close();  } |

问题：

集合映射，映射的集合元素，都是普通的类型， 能否为对象类型？

# 2. 关联映射

需求1：

部门与员工

一个部门有多个员工; 【一对多】

多个员工，属于一个部门 【多对一】

需求2：

项目与开发员工

一个项目，有多个开发人员！

一个开发人员，参与多个项目！ 【多对多】

## 多对一映射与一对多

### 代码

* 需求：员工与部门
* 数据库：
* 设计javabean封装：
* 映射：

|  |
| --- |
| **public** **class** Dept {  **private** **int** deptId;  **private** String deptName;  // 【一对多】 部门对应的多个员工  **private** Set<Employee> emps = **new** HashSet<Employee>(); |
| **public** **class** Employee {  **private** **int** empId;  **private** String empName;  **private** **double** salary;  // 【多对一】员工与部门  **private** Dept dept;  Dept.hbm.xml |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package=*"cn.itcast.b\_one2Many"*>    <class name=*"Dept"* table=*"t\_dept"*>  <id name=*"deptId"*>  <generator class=*"native"*></generator>  </id>  <property name=*"deptName"* length=*"20"*></property>    <!--  一对多关联映射配置 （通过部门管理到员工）  Dept 映射关键点：  1. 指定 映射的集合属性： "emps"  2. 集合属性对应的集合表： "t\_employee"  3. 集合表的外键字段 "t\_employee. dept\_id"  4. 集合元素的类型    -->  <set name=*"emps"*> <!-- table="t\_employee" -->  <key column=*"dept\_id"*></key>  <one-to-many class=*"Employee"*/>  </set>      </class>    </hibernate-mapping> |
|  |
| Employee.hbm.xml |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package=*"cn.itcast.b\_one2Many"*>    <class name=*"Employee"* table=*"t\_employee"*>  <id name=*"empId"*>  <generator class=*"native"*></generator>  </id>  <property name=*"empName"* length=*"20"*></property>  <property name=*"salary"* type=*"double"*></property>    <!--  多对一映射配置  Employee 映射关键点：  1. 映射的部门属性 ： dept  2. 映射的部门属性，对应的外键字段: dept\_id  3. 部门的类型  -->  <many-to-one name=*"dept"* column=*"dept\_id"* class=*"Dept"*></many-to-one>    </class>    </hibernate-mapping> |
| 测试 |
| **public** **class** App {    **private** **static** SessionFactory *sf*;  **static** {  *sf* = **new** Configuration()  .configure()  .addClass(Dept.**class**)  .addClass(Employee.**class**) // 测试时候使用  .buildSessionFactory();  }  // 保存， 部门方 【一的一方法操作】  @Test  **public** **void** save() {    Session session = *sf*.openSession();  session.beginTransaction();    // 部门对象  Dept dept = **new** Dept();  dept.setDeptName("应用开发部");  // 员工对象  Employee emp\_zs = **new** Employee();  emp\_zs.setEmpName("张三");  Employee emp\_ls = **new** Employee();  emp\_ls.setEmpName("李四");  // 关系  dept.getEmps().add(emp\_zs);  dept.getEmps().add(emp\_ls);  // 保存  session.save(emp\_zs);  session.save(emp\_ls);  session.save(dept); // 保存部门，部门下所有的员工    session.getTransaction().commit();  session.close();  /\*  \* 结果  \* Hibernate: insert into t\_employee (empName, salary, dept\_id) values (?, ?, ?)  Hibernate: insert into t\_employee (empName, salary, dept\_id) values (?, ?, ?)  Hibernate: insert into t\_dept (deptName) values (?)  Hibernate: update t\_employee set deptId=? where empId=? 维护员工引用的部门的id  Hibernate: update t\_employee set deptId=? where empId=?  \*/  }  // 【推荐】 保存， 部员方 【多的一方法操作】  @Test  **public** **void** save2() {    Session session = *sf*.openSession();  session.beginTransaction();    // 部门对象  Dept dept = **new** Dept();  dept.setDeptName("综合部");  // 员工对象  Employee emp\_zs = **new** Employee();  emp\_zs.setEmpName("张三");  Employee emp\_ls = **new** Employee();  emp\_ls.setEmpName("李四");  // 关系  emp\_zs.setDept(dept);  emp\_ls.setDept(dept);      // 保存  session.save(dept); // 先保存一的方法  session.save(emp\_zs);  session.save(emp\_ls);// 再保存多的一方，关系回自动维护(映射配置完)    session.getTransaction().commit();  session.close();  /\*  \* 结果  \* Hibernate: insert into t\_dept (deptName) values (?)  Hibernate: insert into t\_employee (empName, salary, dept\_id) values (?, ?, ?)  Hibernate: insert into t\_employee (empName, salary, dept\_id) values (?, ?, ?)  少生成2条update sql  \*/  }    } |

### 总结：

在一对多与多对一的关联关系中，保存数据最好的**通过多的一方来维护关系**，这样可以减少update语句的生成，从而提高hibernate的执行效率！

配置一对多与多对一，这种叫“双向关联”

只配置一对多， 叫“单项一对多”

只配置多对一， 叫“单项多对一”

注意：

配置了哪一方，哪一方才有维护关联关系的权限！

|  |
| --- |
|  |

## Inverse属性 in wos

Inverse属性，是在维护关联关系的时候起作用的。

表示控制权是否转移。(在一的一方起作用)

Inverse , 控制反转。

Inverse = false 不反转； 当前方有控制权

True 控制反转； 当前方没有控制权

维护关联关系中，是否设置inverse属性：

1. 保存数据

**有影响。**

如果设置控制反转,即inverse=true, 然后通过部门方维护关联关系。在保存部门的时候，同时保存员工， 数据会保存，但关联关系不会维护。即外键字段为NULL



2. 获取数据

**无。**

3. 解除关联关系？

**有影响。**

inverse=false， 可以解除关联

inverse=true， 当前方(部门)没有控制权，不能解除关联关系

(不会生成update语句,也不会报错)

4. 删除数据对关联关系的影响？

有影响。

inverse=false, 有控制权， 可以删除。先清空外键引用，再删除数据。

inverse=true, 没有控制权: 如果删除的记录有被外键引用，会报错，违反主外键引用约束！ 如果删除的记录没有被引用，可以直接删除。

## cascade 属性

cascade 表示级联操作 【可以设置到一的一方或多的一方】

none 不级联操作， 默认值

save-update 级联保存或更新

delete 级联删除

save-update,delete 级联保存、更新、删除

all 同上。级联保存、更新、删除

hibernate常见面试题： inverse与cascade区别？

## 多对多映射

需求：项目与开发人员

Project Developer

电商系统

曹吉

王春

OA系统

王春

老张

数据库

### 代码

|  |
| --- |
| /\*\*  \* 开发人员  \*  \* **@author** Jie.Yuan  \*  \*/  **public** **class** Developer {  **private** **int** d\_id;  **private** String d\_name;  // 开发人员，参数的多个项目  **private** Set<Project> projects = **new** HashSet<Project>();  } |
| /\*\*  \* 项目  \*  \* **@author** Jie.Yuan  \*  \*/  **public** **class** Project {  **private** **int** prj\_id;  **private** String prj\_name;  // 项目下的多个员工  **private** Set<Developer> developers = **new** HashSet<Developer>(); |
| Project.hbm.xml |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package=*"cn.itcast.c\_many2many"*>    <class name=*"Project"* table=*"t\_project"*>  <id name=*"prj\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"prj\_name"* length=*"20"*></property>  <!--  多对多映射:  1. 映射的集合属性： “developers”  2. 集合属性，对应的中间表： “t\_relation”  3. 外键字段: prjId  4. 外键字段，对应的中间表字段: did  5. 集合属性元素的类型  -->  <set name=*"developers"* table=*"t\_relation"* cascade=*"save-update"*>  <key column=*"prjId"*></key>  <many-to-many column=*"did"* class=*"Developer"*></many-to-many>  </set>    </class>    </hibernate-mapping> |
|  |
| Developer.hbm.xml |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping package=*"cn.itcast.c\_many2many"*>    <class name=*"Developer"* table=*"t\_developer"*>  <id name=*"d\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"d\_name"* length=*"20"*></property>    <!--  多对多映射配置： 员工方  name 指定映射的集合属性  table 集合属性对应的中间表  key 指定中间表的外键字段(引用当前表t\_developer主键的外键字段)  many-to-many  column 指定外键字段对应的项目字段  class 集合元素的类型  -->  <set name=*"projects"* table=*"t\_relation"*>  <key column=*"did"*></key>  <many-to-many column=*"prjId"* class=*"Project"*></many-to-many>  </set>      </class>    </hibernate-mapping> |

### 维护关联关系

设置inverse属性，在多对多种维护关联关系的影响？

1） 保存数据

有影响。

inverse=false ，有控制权，可以维护关联关系； 保存数据的时候会把对象关系插入中间表；

inverse=true, 没有控制权， 不会往中间表插入数据。

2） 获取数据

无。

3） 解除关系

// 有影响。

// inverse=false ,有控制权， 解除关系就是删除中间表的数据。

// inverse=true, 没有控制权，不能解除关系。

4） 删除数据

有影响。

// inverse=false, 有控制权。 先删除中间表数据，再删除自身。

// inverse=true, 没有控制权。 如果删除的数据有被引用，会报错！ 否则，才可以删除

需求1：

老师与学员，

一个老师，对应多个学员

一个学员，有多个老师教

需求2：

父亲/孩子

学生与考试成绩

….

实现：

1. 了解需求

2. 设计数据库 （数据库结构要先出来）

3. 写javabean 、 映射 (生成数据库)

4. 测试：

保存、获取、解除关系、删除数据

(inverse属性)